Structured Case-Based Reasoning for Inference-Time Adaptation of Text-to-SQL Parsers

Abhijeet Awasthi, Soumen Chakrabarti, Sunita Sarawagi

Department of Computer Science and Engineering
Indian Institute of Technology Bombay, Mumbai, India
{awasthi,soumen,sunita}@cse.iitb.ac.in

Abstract

Inference-time adaptation methods for semantic parsing are useful for leveraging examples from newly-observed domains without repeated fine-tuning. Existing approaches typically bias the decoder by simply concatenating input-output example pairs (cases) from the new domain at the encoder's input in a Seq-to-Seq model. Such methods cannot adequately leverage the structure of logical forms in the case examples. We propose StructCBR, a structured case-based reasoning approach, which leverages subtree-level similarity between logical forms of cases and candidate outputs, resulting in better decoder decisions. For the task of adapting Text-to-SQL models to unseen schemas, we show that exploiting case examples in a structured manner via StructCBR offers consistent performance improvements over prior inference-time adaptation methods across five different databases. To the best of our knowledge, we are the first to attempt inference-time adaptation of Text-to-SQL models, and harness trainable structured similarity between subqueries.

Introduction

Natural language interfaces to databases (Zelle and Mooney 1996; Tang and Mooney 2000; Popescu, Etzioni, and Kautz 2003) enable access to structured information for users who are not familiar with languages like SQL by parsing user provided text-queries into executable SQLs. Text-to-SQL semantic parsing is a challenging task that not only demands robust natural language understanding but simultaneously requires reasoning over the schema structure of the databases. Databases containing similar information (e.g. census in various countries) may be designed using diverse schema structures, thus making it hard for the model to generalize across schemas unseen during training. Hence, Text-to-SQL models often struggle to parse text queries for a new schema in a zero-shot manner (Suhr et al. 2020; Lee, Polozov, and Richardson 2021; Hazoom, Malik, and Bogin 2021). In practice, a small number of Text-to-SQL examples in the target schema are often essential for successful model adaptation. However, finetuning a Text-to-SQL model for each new database is not generally practical, for the following reasons: (i) Huge variation in database schema makes it tedious to collect sufficiently large finetuning datasets for each schema, while finetuning on small datasets is unavoidably fraught with over–fitting, catastrophic forgetting, and instability w.r.t. random seeds. (ii) Finetuning may take considerable time, preventing fast incorporation of new data into the model. (iii) Often, a single large-footprint model serves multiple clients with diverse databases at the same time. Fine-tuning a separate model for each database is considered too resource-intensive in such multi-tenant scenarios.

Therefore, we focus on fast online adaptation of Text-to-SQL models without parameter updates, until the next cycle of finetuning is deemed feasible. Recently, case-based reasoning (CBR), which utilizes a memory of past labeled examples as cases, has emerged as a promising paradigm of inference-time adaptation without finetuning (Das et al. 2020, 2021; Pasupat, Zhang, and Guu 2021; Gupta et al. 2021). CBR has been found effective for tasks like knowledge graph completion (KGC) (Das et al. 2020), question answering over knowledge bases (KBQA) (Das et al. 2021), task-oriented semantic parsing (Pasupat, Zhang, and Guu 2021; Gupta et al. 2021), translation (Khandelwal et al. 2021), and text-based games (Atzeni et al. 2022). However, many prior CBR approaches designed around Seq2Seq architectures simply concatenate input-output cases with the current input at the encoder (Das et al. 2021; Pasupat, Zhang, and Guu 2021; Gupta et al. 2021). These methods do not leverage the structure of logical forms (query plan trees) in case examples.

In response, we propose StructCBR, a *structured* CBR approach that directly exploits sub-tree level similarities between the candidate outputs and the case examples for adapting a Text-to-SQL model to a new schema. We start with SmBoP (Rubin and Berant 2021), a recent semi-auto-regressive architecture that decodes query trees bottom-up, respecting the structure of SQL grammar production rules, instead of left-to-right token-level decoding in Seq2Seq models (Guo et al. 2019; Wang et al. 2020; Scholak et al. 2021; Scholak, Schucher, and Bahdanau 2021). We implement a novel *structured case memory lookup* module to boost scores of promising candidate trees using sub-tree level similarity with case trees under similar input context. This similarity is trainable. We show that explicitly-learned structured memory lookup leads to more accurate transfer from cases, compared to prior inference-time adaptation methods such as ConcatCBR and

GTM (Khandelwal et al. 2020; Das et al. 2021; Khandelwal et al. 2021) that we implemented both on SmBoP, and other Seq2Seq Text-to-SQL architectures like T5-large.

We summarize our contributions as follows:

1) We propose StructCBR, which, to our knowledge, is the first inference-time adaptation method for Text-to-SQL parsing without parameter fine-tuning.

2) StructCBR incorporates a novel structured case memory and trainable query subtree similarity module that can boost scores of likely-correct outputs during inference. This is in contrast with earlier approaches like ConcatCBR and GTM.

3) We propose a trainable compositional sub-tree similarity function that is both more accurate and more efficient for scoring large search frontiers, compared to default whole-tree embeddings.

4) Through experiments with five database schemas (§ 14) of varying complexity, we observe that StructCBR is consistently better than prior inference-time adaptation methods on both SmBoP and sequence-based Text-to-SQL models.

5) We show that StructCBR provides almost instant adaptation to a target schema. In contrast, finetuning (§ 14) can be up to 500 times slower.

SmBoP Preliminaries

We present a brief background on SmBoP here. Readers familiar with SmBoP can largely skip this section. SmBoP converts a natural language question $\bar{x} \in \mathcal{X}$ (called the 'utterance') targeting a database schema $\bar{s} \in \mathcal{S}$, to an SQL query $\hat{q} \in \mathcal{Q}$. We describe the major modules in SmBoP.

Utterance and schema encoding: Given token sequence $\bar{x} = [x_1, x_2, \dots, x_n]$ in the text query, and database schema $\bar{s} = [s_1, s_2, \dots, s_m]$ denoting table and column names, SmBoP jointly encodes them using a pre-trained Transformer like RoBERTa (Liu et al. 2019) followed by relation-aware-transformer (RAT) layers (Shaw, Uszkoreit, and Vaswani 2018; Wang et al. 2020; Scholak et al. 2021). We denote the output from the last encoder layer as $\bar{\mathbf{x}} = [\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_n]$ and $\bar{\mathbf{s}} = [\mathbf{s}_1, \mathbf{s}_2, \dots \mathbf{s}_m]$, representing the jointly encoded contextual embeddings of text tokens and schema elements respectively.

Decoding SQL output: Unlike standard sequence-based decoding (Wang et al. 2020; Scholak et al. 2021; Scholak, Schucher, and Bahdanau 2021), SmBoP decodes the SQL tree bottom-up and in layers. SmBoP views any SQL query as a height-balanced relational algebra tree converted using a special idempotent KEEP operator κ as shown in Figure 1. Given a beam size K, at decoding step t, the decoder beam B_t comprises K candidate sub-trees of height t from the bottom. At step $t+1$, trees from B_t are grown either via unary operators (e.g. COUNT), or by combining two trees in B_t using a binary operator (e.g. $>$), as per the SQL grammar. The candidate trees at step $t+1$ form a frontier set F_{t+1} and is of size $|F_{t+1}| = K^2|\mathcal{B}| + K|\mathcal{U}|$, where $\mathcal{B}$ and $\mathcal{U}$ represent the set of binary and unary operations respectively. SmBoP assigns each candidate tree $z \in F_{t+1}$ a score $s_\theta(z)$ (described below). The top-K highest scoring trees in F_{t+1} form the next beam B_{t+1}. This continues up to a maximum height T, when the highest scoring tree in B_T is output.

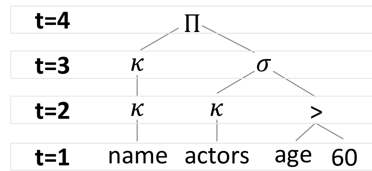

Figure 1: SmBoP (Rubin and Berant 2021) decodes SQL as a balanced relational algebra tree. At each level t, trees in the beam combine via unary or binary operators to form candidates of the next beam. StructCBR leverages CBR on generated sub-trees.

Text 1	*Give the code of the airport with the fewest number of flights*
SmBoP output	`SELECT sourceairport FROM flights GROUP BY sourceairport ORDER BY SUM(flightno) ASC LIMIT 1`
Correct SQL	`SELECT airportcode FROM airports JOIN flights ON airportcode = sourceairport GROUP BY sourceairport ORDER BY COUNT(*) ASC LIMIT 1`
Text 2	*What is the code of the airport that has the highest number of flights?*

Table 1: Illustration of the lack of generalization of Text-to-SQL to new schema.

Scoring a tree: A tree $z = (z_b, z_\ell, z_r)$ consists of root operator z_b and subtrees z_ℓ, z_r. SmBoP encodes a variable-size tree z into two fixed dimensional vectors: (i) $\mathbf{z}$: an embedding of the tree computed recursively on the tree structure, where a transformer outputs $\mathbf{z} = \mathrm{TX}_\theta([z_b, \mathbf{z}_\ell, \mathbf{z}_r])$; (ii) $\mathbf{z}'$: a contextual representation of z grounded in input text $\bar{\mathbf{x}}$ computed via a multiheaded cross attention module $\mathbf{z}' = \mathrm{XAtt}_\theta(\mathbf{z}, \bar{\mathbf{x}})$. SmBoP computes the score of a tree $z \in F_{t+1}$, as follows:

$$s_\theta(z) = \mathbf{w}_{z_b}^T \, \mathrm{FF}_\theta([\mathbf{z}_\ell; \mathbf{z}'_\ell; \mathbf{z}_r; \mathbf{z}'_r]) \quad (1)$$

where FF_θ is a feed forward network, and $\mathbf{w}_{z_b}$ represents a learned embedding of operator z_b.

The model is trained using Text-SQL pairs from a set of training schema to maximize the likelihood of the correct subtrees at each beam. During inference, when presented with text utterances relating to a new database schema, the model often fails to discover the mapping of the text to schema names and relationships in the new schema. Table 1 presents an example where a SmBoP model trained on the Spider dataset (Yu et al. 2018) is deployed on a new schema about `flights`. On inspecting the predicted and correct SQL, we find that the model failed to reason that *number of flights* requires a `count(*)` instead of `sum(flightno)`. Now suppose an expert provides the correct SQL as additional information to be used during inference of subsequent queries. Consider a second query (shown as Text 2 in Table 1) that also needs to reason about *number of flights*, and the default SmBoP makes similar errors (not shown). Only existing mechanism in SmBoP is to fine-tune parameters which could be time-consuming and unstable. In the next section we show how our method can instantaneously leverage test-time user labels to predict the correct SQL for Text 2. More such anecdotes appear in Table A4 of the Appendix.

Proposed Method: StructCBR

We aim to learn a Text-to-SQL model M, using a dataset $\mathcal{D}_{\text{train}}$ of Text-SQL pairs such that it is capable of **C1:** Independently translating the text queries $\bar{x}$ to executable SQL programs $\hat{q}$, and **C2:** Utilizing a small set $\mathcal{D}_{\text{new}}$ of Text-SQL pairs from a target schema $\mathbf{s}_{\text{new}}$, to improve its own predictions during inference, without finetuning. In line with prior work (Das et al. 2020, 2021), we refer to the second capability **C2** as Case-based reasoning (CBR), and the dataset $\mathcal{D}_{\text{new}}$ of Text-SQL pairs in the target schema as cases.

The StructCBR module leverages the similarity between gold subtrees that appear in similar contexts in the set of cases $\mathcal{D}_{\text{new}}$ and the candidate subtrees in SmBoP's frontier F_{t+1}, to boost the scores of likely-correct candidates at each decoding step $t+1$. Consider a subtree z in the frontier F_{t+1} for an input text $\bar{x}$, a case-example with text question as $\bar{x}^c$, and the gold SQL tree as $\mathcal{Z}^c_{\text{gold}}$. Let z^c be a subtree of $\mathcal{Z}^c_{\text{gold}}$. The key idea of StructCBR is, if z and z^c are structurally similar, and appear in similar contexts w.r.t. $\bar{x}$ and $\bar{x}^c$, then there is a strong evidence that the subtree z should also appear as a part of the gold tree $\mathcal{Z}_{\text{gold}}$ of $\bar{x}$. Figure 2 provides an illustration with $z = \text{age} > 60$ in the candidate frontier F_{t+1}, and a similarly structured case tree $z^c = \text{age} > 80$ appearing in a similar context $\bar{x}^c$ (both contain the phrase `who are past`).

Even though the key idea of matching with case sub-trees is simple, several important design choices had to be made to ensure that CBR inter-operates efficiently with SmBoP's own scoring, and consistently improves its performance in the presence of multiple cases of varying levels of relatedness. First, how should we compute the *contextual* similarity of a candidate tree z with a case tree, given that memory would also contain unrelated cases that would match wrong candidate trees? Second, how can we efficiently compute the similarity of all candidate trees with all entries in the case memory? Unlike Seq2Seq models that do not perform beam-search during training, SmBoP generates a large search frontier even during training. We elaborate on how our design tackles these challenges next.

Algorithm 1 presents the high-level pseudo code of StructCBR with additions to the SmBoP model.

Choosing Tree Representations

We need to choose a representation of a tree z using which we can efficiently compute similarity with case trees. Just the structural similarity of z with a case z^c is not sufficient unless we also contextualize them on their respective inputs. Accordingly, we design an embedding function $G_\phi(z, \bar{x}) \mapsto \mathbb{R}^d$ that jointly encodes a candidate tree z corresponding to an input $\bar{x}$ as a d dimensional vector. We train a separate

Algorithm 1: SmBoP with StructCBR.

input: $\bar{x}$, $\bar{s}$, $\mathcal{D}_{\text{new}}$
$M \leftarrow \text{CreateCaseMemory}(\mathcal{D}_{\text{new}})$ (§ 14)
$\bar{\mathbf{x}}, \bar{\mathbf{s}} \leftarrow \text{EncodeTextSchema}_\theta(\bar{x}, \bar{s})$
$B_0 \leftarrow$ top-Kschema constants and DB values
for $t \leftarrow 0 \ldots T-1$ **do**
  $\mathbf{z} \leftarrow \text{CreateTreeReps}(z)$
  $\mathbf{z}' \leftarrow \text{GroundTreeReps}(\mathbf{z}, \mathbf{x})$
  $p_\theta \leftarrow \text{SmBoPScores}(\mathbf{z}, \mathbf{z}')$ (§)
  $G_\phi(z, \bar{x}) \leftarrow \text{JointReps}(\mathbf{z}, \mathbf{z}', \mathbf{x})$ (Eqn 2)
  $\text{sim}_\phi \leftarrow \text{TreeSim}(G_\phi(z, \bar{x}), M)$ (Eqn 4)
  $p_\phi \leftarrow \text{StructCBRScores}(\text{sim}_\phi, M)$ (Eqn 5)
  $F_{t+1} \leftarrow \text{CombineScores}(p_\theta, p_\phi)$ (Eqn 6)
  $B_{t+1} \leftarrow \text{top-}K(F_{t+1})$
return $\text{argmax}_z(B_T)$

transformer model TX_ϕ with parameters ϕ that takes as input four vectors: $\mathbf{z}$ that encodes the structure of the tree z, $\mathbf{z}'$ that is the contextual representation of z defined in § , an embedding $\mathbf{w}_b$ of z's root node b, and $\text{pool}(\mathbf{x})$ a mean-pooled version of the input text representation $\bar{\mathbf{x}}$:

$$G_\phi(z, \bar{x}) = \text{TX}_\phi([\mathbf{z}, \mathbf{z}', \mathbf{w}_b, \text{pool}(\mathbf{x})]). \quad (2)$$

This embedding captures both the structure and context and the parameters ϕ are trained to co-embed similar trees in matching contexts, while pulling apart pairs differing either structurally or contextually. For example, in Figure 2 if the query text was `Name all actors who are 60 or above`, then the similarity of candidate `age > 60` from the same case sub-tree should be reduced. Unlike recursive tree representations (Socher et al. 2013), here contextualization w.r.t. $\bar{x}$ plays a critical role.

Case Memory Design

We construct a case memory M over the gold SQL trees $\{\mathcal{Z}^c_{\text{gold}}\}$ for *all* cases in $\mathcal{D}_{\text{new}}$. Corresponding to each node b of a gold tree $\mathcal{Z}^c_{\text{gold}}$ we form a subtree rooted at b and including the part of $\mathcal{Z}^c_{\text{gold}}$ below b. Thus, the size of the case memory is the total number of nodes over all gold trees in cases. The encoding $G_\phi(z^c, \bar{x}^c)$ of each subtree z^c for a case $(\bar{x}^c, \mathcal{Z}^c_{\text{gold}})$ in $\mathcal{D}_{\text{new}}$ is pre-computed using Equation 2 and stored in M.

Efficient Tree Similarity Computation

We need to compute the similarity of each tree z in the frontier F_{t+1} with all case sub-trees $z^c \in M$. One way to compute similarity between trees z and z^c is based on ℓ_2 distance[1] between their G_ϕ representations as follows:

$$\text{sim}_\phi(z, z^c, \bar{x}, \bar{x}^c) = -\left\|G_\phi(z, \bar{x}) - G_\phi(z^c, \bar{x}^c)\right\|_2 \quad (3)$$

However, computing G_ϕ representations for each tree $z \in F_{t+1}$ entails large memory and compute costs since the frontier size $|F_{t+1}| = K^2|\mathcal{B}| + K|\mathcal{U}|$ is quadratic in beam-size K. With the default K for SmBoP being 30, and size of

[1]Like Khandelwal et al. (2020) we observed better results with ℓ_2 distance, in comparison to inner product.

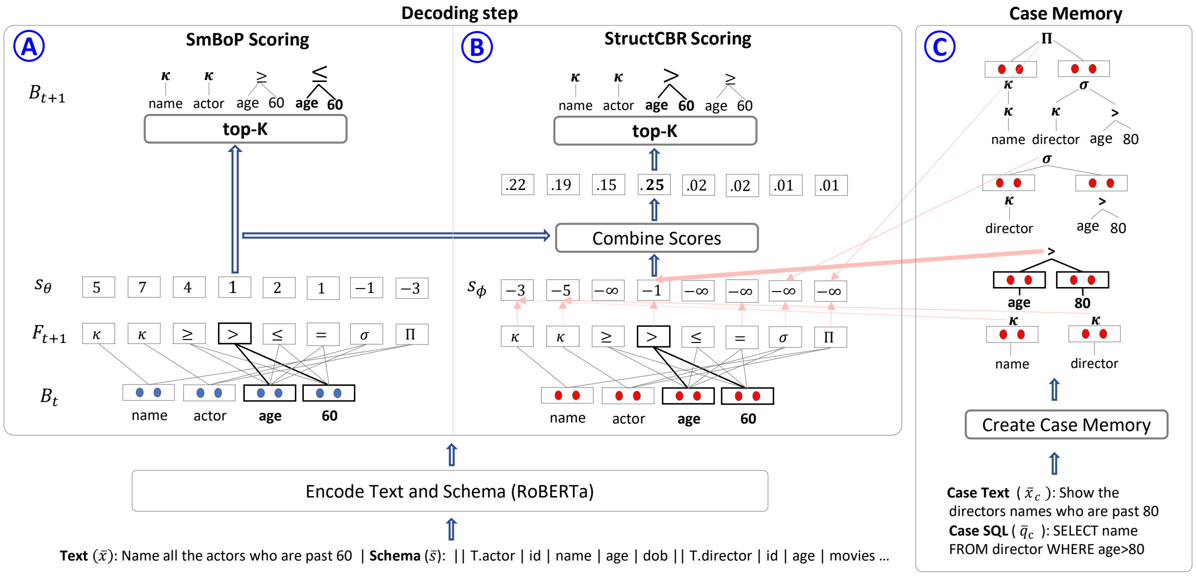

Figure 2: Augmenting SmBoP with StructCBR (Structured Case-based Reasoning): In part Ⓐ, the top-K step in SmBoP scoring misses the correct sub-tree `age > 60` due to a lower score (score=1) w.r.t. competing sub-trees in the frontier F_{t+1} like `age ≥ 60` (score=4) and `age ≤ 60` (score=2). In part Ⓒ, StructCBR creates a memory of all the sub-tree representations available in cases as described in § 14. In part Ⓑ, StructCBR scores the frontier candidates based on learned tree-similarities w.r.t. the sub-trees in cases as described in § 14 and § 14. For example, StructCBR boosts the score of `age > 60` because of its high similarity with the case sub-tree `age > 80` and similarity of context `who are past`. Thus, the top-K step applied on the combined SmBoP and StructCBR scores recovers the correct sub-trees that otherwise may get missed based on SmBoP's scoring alone. For brevity, we consider only one case-example in this figure.

the SmBoP grammar, this translates to around 23 thousand trees per frontier. Pruning the frontier F_{t+1} based on SmBoP scores alone resulted in poor performance. This led us to design an alternative compositional CBR scoring method that can more efficiently score all candidate trees in the frontier.

Our key idea is to compute the similarity between two trees compositionally as a function of similarity between their left and right children respectively. This requires only $\mathcal{O}(K)$ tree representations for the trees in beam B_t as against $K^2|\mathcal{B}| + K|\mathcal{U}|$ operations of the whole-tree approach. Recall that the trees $z \in F_{t+1}$ are formed by combining trees in beam B_t via SQL operations. A tree $z \in F_{t+1}$ can thus be represented as $z = (z_b, z_\ell, z_r)$ where z_ℓ, $z_r \in B_t$ denote the left and right child respectively, and z_b is the root node combining z_ℓ and z_r. After the beam B_t is created, we compute the embedding G_ϕ for each tree in B_t using Equation (2). Now, the similarity between a candidate tree $z = (z_b, z_\ell, z_r)$ for an input text $\bar{x}$, and a case sub-tree $z^c = (z^c_b, z^c_\ell, z^c_r)$ on input text $\bar{x}^c$ in memory M is computed as:

$$\widehat{\text{sim}}_\phi(z, z^c, \bar{x}, \bar{x}^c) = \text{sim}_\phi(z_\ell, z^c_\ell, \bar{x}, \bar{x}^c) + \text{sim}_\phi(z_r, z^c_r, \bar{x}, \bar{x}^c) \quad (4)$$

In Section 14, we show that using this similarity function provides better results by allowing the entire frontier to be scored more efficiently in comparison to computing similarities based on Equation 3 only for a subset of trees in the frontier pruned based on SmBoP scores.

Boosting SmBoP Frontier With Tree Similarities

To compute an overall score of a candidate tree $z \in F_{t+1}$ based on its similarity with the case sub-trees in M, we aggregate over all the case sub-trees z^c with the same root node ($z_b = z^c_b$) using a logsumexp operator, which provides us a soft-maxed similarity of z w.r.t. case sub-trees.

$$s_\phi(z) = \log \sum_{c \in M \wedge z_b = z^c_b} \exp(\widehat{\text{sim}}_\phi(z, z^c, \bar{x}, \bar{x}^c)) \quad (5)$$

Now every candidate tree $z \in F_{t+1}$ has two scores: $s_\theta(z)$ assigned by default SmBoP and $s_\phi(z)$ computed by StructCBR. The scores $s_\theta(z)$ and $s_\phi(z)$ can lie in very different ranges. Summing them in a normalized probability space provided better results than summing the scores directly. Hence, we independently normalize $s_\theta(z)$ to $p_\theta(z)$ and $s_\phi(z)$ to $p_\phi(z)$ by a softmax operation applied over all trees in the frontier. The combined score of a frontier tree z is:

$$p(z) = (p_\theta(z) + p_\phi(z))/2. \quad (6)$$

Supervising StructCBR

During training, we assume availability of training data $\mathcal{D}_{\text{train}} = \{(\bar{x}_i, \bar{s}_i, \bar{q}_i)\}_{i=1}^N$ consisting of utterances $\bar{x}_i$ on a schema $\bar{s}_i$, and the corresponding gold SQL queries $\bar{q}_i$. We first train the SmBoP model, parameterized as θ, using $\mathcal{D}_{\text{train}}$. The training objective of SmBoP for a single example maximizes the likelihood of sub-trees that are part of the tree $\mathcal{Z}_{\text{gold}}$

corresponding to gold SQL $\bar{q}$:

$$\mathcal{L}_\theta = -\sum_{t=0}^{T} \sum_{z_t \in \mathcal{Z}_{\text{gold}}} \log p_\theta(z_t). \tag{7}$$

Next, we introduce the StructCBR module parameterized as ϕ on top of the (now frozen) SmBoP model. We observed training the StructCBR parameters ϕ while freezing the learned SmBoP parameters θ to provide slightly better results in comparison to training both θ and ϕ jointly. The parameters ϕ are also learned using $\mathcal{D}_{\text{train}}$ by maximizing the likelihood of the gold subtrees as per the distributions p_ϕ and p through the following loss function:

$$\mathcal{L}_\phi = -\sum_{t=0}^{T} \sum_{z_t \in \mathcal{Z}_{\text{gold}}} \log p_\phi(z_t) + \log p(z_t) \tag{8}$$

The $-\log p_\phi(z_t)$ term maximizes the likelihood of gold trees w.r.t. the CBR distribution p_ϕ, independent of the SmBoP distribution p_θ. Similarly, the $-\log p(z_t)$ term maximize the likelihood of the gold trees w.r.t. the combined distribution p (Eqn 6). During training, we design each training batch to contain C examples from same schema so that for a given train example, the remaining $C-1$ examples serve as the cases from the same schema. We train with $C = 32$ and a batch-size of 64.

Related Work

We review prior work on inference-time model adaptation for related tasks and also describe our adaptation of some of these works in the context of Text-to-SQL for comparisons with StructCBR.

Concatenating related examples with input: A common approach, that we call ConcatCBR, for utilizing cases during inference is to concatenate the input-output pair of each case along with the input text at the encoder of a Seq2Seq model. During training, the decoder is expected to learn to utilize the cases on the encoder side. Das et al. (2021) utilize ConcatCBR for question answering over knowledge bases, and Pasupat, Zhang, and Guu (2021); Gupta et al. (2021) utilize ConcatCBR for other semantic parsing tasks. ConcatCBR is similar to the retrieve and edit framework for structured outputs (Hashimoto et al. 2018) and machine translation (Hossain, Ghazvininejad, and Zettlemoyer 2020). For the Text-to-SQL task, we implement a ConcatCBR baseline that trains an SmBoP model to use retrieved Text-SQL examples concatenated with the input-text. During inference, the retrieval index is updated with the case-examples from the target schema.

Generalization through Memorization (GTM): Khandelwal et al. (2020, 2021) propose a memory look-up based method for adapting pre-trained language and machine translation models to a target domain. Given a target dataset, their method constructs a look-up index by using contextual embeddings from the pre-trained model as keys and the corresponding text tokens as values. During inference the model scores are interpolated with the similarity scores aggregated over the nearest neighbours in the loop-up index. For our Text-to-SQL set-up, we implement this baseline using a trained SmBoP model. We memorize the dataset $\mathcal{D}_{\text{new}}$ in the target schema by creating a look-up index with embeddings of child subtrees from SmBoP as keys: $[\mathbf{z}_\ell; \mathbf{z}'_\ell; \mathbf{z}_r; \mathbf{z}'_r]$, and their parent nodes as values. During inference, the scores from the SmBoP model are interpolated with neighbour similarities in a way similar to Khandelwal et al. (2021). Unlike StructCBR and ConcatCBR, this baseline (GTM) does not explicitly train the SmBoP model for utilizing the cases during inference.

We discuss other related work in Appendix.

Experiments[2]

We evaluate StructCBR for adapting a Text-to-SQL model to five different target schemas without finetuning. The target schemas are chosen from varying domains. We compare StructCBR with prior inference-time adaptation methods discussed in § 14, and present an ablation study. We also show that StructCBR enables much faster adaptation of Text-to-SQL models in comparison to finetuning.

Datasets: We utilize Spider (Yu et al. 2018), which is a collection of Text-to-SQL examples covering 200 unique schemas. We use the train split of Spider as $\mathcal{D}_{\text{train}}$, for training all the models. $\mathcal{D}_{\text{train}}$ contains 7000 Text-SQL example pairs from 140 databases. For evaluation, we hold out the following five databases containing the most examples from Spider's dev set [3]: {world_1, car_1, cre_Doc_Template_Mgt, dog_kennels, flight_2}. The five evaluation databases do not occur in the train set, and belong to sufficiently different domains of varying difficulty. The remaining part of the dev set containing 576 examples is used for model selection while training on $\mathcal{D}_{\text{train}}$. We hold out 30 randomly selected examples from each of the five selected databases as $\mathcal{D}_{\text{new}}$ (cases) for adaptation, and use the remaining examples as the test set, $\mathcal{D}_{\text{test}}$. The average size of $\mathcal{D}_{\text{test}}$ is 60, and varies from roughly 50 to 90 examples across the five schemas. To ensure robust evaluation, we report numbers averaged over three random $\mathcal{D}_{\text{new}}/\mathcal{D}_{\text{test}}$ splits. We also report the numbers micro-averaged over all the 300 test examples across the five schemas.

Evaluation metrics: Following prior work (Yu et al. 2018), we report Execution Accuracy **(EX)** and Exact-Set-Match Accuracy **(EM)** for all the methods. EX returns 1 if executing the gold query $\bar{q}$ and the predicted query $\hat{q}$ on the target database gives the same results. EM compares all the SQL clauses within $\bar{q}$ and $\hat{q}$ and returns 1 if all the clauses match, except possibly the DB-values (constants) in the SQL query. Most Text-to-SQL models utilize beam search, and return the top-K highest scoring candidates in the beam as the output. Hence, we also report the top-K versions of EM and EX metrics as BEM and BEX respectively, where K is the beam size. In our experiments, $K = 30$. BEM/BEX for a beam is 1, if at least one of the candidates in the beam has an EM/EX of 1.

[2]Code: https://github.com/awasthiabhijeet/structcbr

[3]Spider's test set is publicly inaccessible as of 08/15/2022.

Methods compared: We compare the accuracy of StructCBR after adaptation with the following methods: **(i)** SmBoP: The base model without any adaptation to benchmark the gains from different inference-time adaptation methods. **(ii)** ConcatCBR: The standard method of concatenating input-output case examples with the input-text. **(iii)** GTM: Mapping $\mathcal{D}_{\text{new}}$ using SmBoP into a non-parametric memory for augmenting model's predictions with inference-time memory look-ups similar to Khandelwal et al. (2020, 2021). We discussed ConcatCBR and GTM baselines in Section 14. All the baselines are implemented using SmBoP as the base model. In Appendix we also present ConcatCBR implemented on a T5-based Seq2Seq model.

Implementation details: We implemented StructCBR and baselines using AllenNLP (Gardner et al. 2018) and Transformers (Wolf et al. 2020) libraries. We utilize the authors' implementation of SmBoP (Rubin and Berant 2021). Due to limited computing resources, we primarily experiment with the ROBERTA-BASE checkpoint for initializing the text encoder, followed by four RAT layers (Wang et al. 2020) to encode the schema structure. All other hyper-parameters are the set to their default values. The SmBoP model is trained on $\mathcal{D}_{\text{train}}$ for 60K steps with a batch size of 80, using the default learning rate (LR) of 1.86×10^{-4}. The GTM baseline utilizes the output of this model for memory look-ups. For ConcatCBR baseline we train the SmBoP model further for 60K steps with a LR of 5×10^{-5}, while concatenating the retrieved cases in the encoder's input. StructCBR introduces 2.53% additional parameters (ϕ) over the SmBoP parameters (θ). We train the parameters ϕ on $\mathcal{D}_{\text{train}}$ using a batch size of 64 for 60K steps with the default LR of 1.86×10^{-4}. Additional training details are provided in Appendix.

Overall Results: In Table 2, we compare StructCBR with different inference-time methods for adapting SmBoP based models on five different evaluation schemas. The performance of the unadapted SmBoP model varies from 46.1 EM to 84.3 EM across the five schemas indicating that the evaluation schemas are of varying difficulty. We find that StructCBR almost consistently offers substantial gains over the base SmBoP model w.r.t. all the metrics, and across all the five schemas. StructCBR gains upto 6.3 EM points and on average 4.6 EM points over the base SmBoP model, while achieving almost 4 times higher gains than best existing method. In contrast, the ConcatCBR method, which has been shown to work well for other semantic parsing tasks (Das et al. 2021; Pasupat, Zhang, and Guu 2021), provides positive EM gains for only three out of five schemas, and causes overall drop in micro-averaged EM over all the test instances. We also explored a ConcatCBR implementation based on T5-large model with a comparable base accuracy (Appendix). Even compared to this method, we continue to observe higher and more consistent gains from StructCBR on SmBoP. GTM, another inference-time adaptation baseline, utilizes memory lookups similar to Khandelwal et al. (2021), and offers almost consistent but much smaller gains in comparison to StructCBR. The GTM baseline performs memory look-ups based on representations learned by the base SmBoP model whereas StructCBR is explicitly trained to perform sub-tree level memory look-ups. In particular, StructCBR boosts the top-K EM score (BEM) by up to 12.3 points. With a large-sized SmBoP architecture we continue to observe consistent gains for most of the schemas. Appendix provide more results for adapting the large-sized models in the same setting as Table 2, and some anecdotal examples that were originally mispredicted by the base SmBoP model, but were fixed by StructCBR.

	EM	EX	BEM	BEX
$\mathbf{s}_{\text{new}}$ = world_1 , $\lvert\mathcal{D}_{\text{test}}\rvert = 89$				
SmBOP	46.1	36.3	55.4	49.0
ConcatCBR	-4.1	-4.9	-5.2	-6.3
GTM	+0.0	+0.0	+0.4	-0.4
StructCBR (Ours)	**+2.6**	**+2.6**	**+3.8**	+0.0
$\mathbf{s}_{\text{new}}$ = car_1 , $\lvert\mathcal{D}_{\text{test}}\rvert = 60$				
SmBOP	43.3	45.6	50.6	53.9
ConcatCBR	+3.9	+3.9	+4.4	+3.4
GTM	+2.2	+2.2	+3.9	+2.8
StructCBR (Ours)	**+6.1**	**+6.7**	**+8.3**	**+7.2**
$\mathbf{s}_{\text{new}}$ = cre_Doc_Template_Mgt , $\lvert\mathcal{D}_{\text{test}}\rvert = 53$				
SmBOP	84.3	89.3	94.3	96.2
ConcatCBR	+5.7	**+5.0**	+1.9	+1.3
GTM	+3.2	+1.9	+1.9	+1.3
StructCBR (Ours)	**+6.3**	+3.8	**+3.8**	**+2.5**
$\mathbf{s}_{\text{new}}$ = dog_kennels , $\lvert\mathcal{D}_{\text{test}}\rvert = 49$				
SmBOP	66.6	59.2	80.3	74.8
ConcatCBR	+0.7	+0.0	-2.0	-1.3
GTM	+1.4	+2.7	+0.7	+3.4
StructCBR (Ours)	**+3.4**	**+6.1**	**+3.4**	**+4.1**
$\mathbf{s}_{\text{new}}$ = flight_2 , $\lvert\mathcal{D}_{\text{test}}\rvert = 49$				
SmBOP	55.8	67.4	59.2	80.3
ConcatCBR	-8.8	-6.1	-2.0	-1.3
GTM	+0.0	-1.4	+2.1	+0.0
StructCBR (Ours)	**+5.5**	**+4.1**	**+12.3**	**+4.8**
Micro-Average , $\lvert\mathcal{D}_{\text{test}}\rvert = 300$				
SmBOP	57.2	56.3	66.0	67.6
ConcatCBR	-0.8	-0.8	-1.0	-1.4
GTM	+1.2	+1.0	+1.7	+1.2
StructCBR (Ours)	**+4.6**	**+4.4**	**+6.0**	**+3.4**

Table 2: Comparison of StructCBR with prior inference time adaptation methods (ConcatCBR and GTM) on 5 different schemas. SmBoP row provides the performance of the unadapted model, while other rows report gains over SmBoP after adaptation. Micro-average refers to numbers averaged over all the test instances spanning across the five schemas. $\lvert\mathcal{D}_{\text{test}}\rvert$ refers to size of the test-set, and $\mathbf{s}_{\text{new}}$ provides the schema name.

Impact of number of cases: We run another set of experiments by reducing the number of cases from 30 to 20. Figure 3 shows that gains from both GTM and StructCBR are smaller with fewer cases, as expected. Interestingly, Struct-

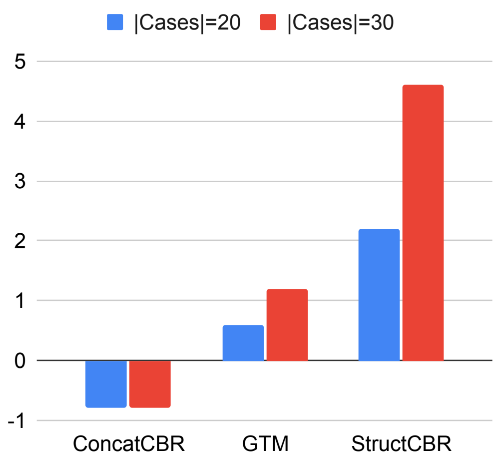

Figure 3: Impact of case size on gains in EM (y-axis) of adapted models (x-axis). With 20 cases, StructCBR still outperforms GTM with 30 case examples.

CBR with 20 cases still outperforms GTM with 30 case examples. For ConcatCBR we do not observe significant changes because it additionally augments the case memory with examples retrieved from Spider's train set. Not augmenting cases via retrieval resulted in even worse performance of ConcatCBR.

Justification for tree similarity: In Section 14, we argued that directly computing tree similarities (sim_ϕ) using Equation (3) was inefficient, and required pruning to be practical. Instead in StructCBR we compute tree similarity ($\widehat{\text{sim}_\phi}$ as per Equation 4) more efficiently as a composition of similarity of its children, and does not require pruning. Table 3 justifies our design by comparing results of scoring a *pruned frontier* containing top-$5K$ trees using sim_ϕ, with scoring the *entire frontier* using $\widehat{\text{sim}}_\phi$. Efficiently scoring the entire frontier provides us better results on 4 out 5 schemas and a micro-averaged gain of 2.2 points in EM.

	world_1	car_1	cre _Doc	dog _kenn	flights_2	Micro avg
Whole-tree	43.4	48.9	88.7	**76.2**	55.1	59.6
Ours	**48.7**	**49.4**	**90.6**	70.1	**61.2**	**61.8**

Table 3: Scoring *pruned frontier* as per similarity of whole trees as in Equation (3) vs. our scoring of the *entire frontier* as per similarity composed from subtrees, as in Equation (4). Better performance on four out of five schemas justifies our design of tree similarity.

Comparison with fine-tuning: Adapting transformer models via finetuning often provides significant accuracy improvements, thanks to recent advances in optimization of transformer models (Huang et al. 2020; Xu et al. 2021). However, finetuning is not viable when a model needs to be adapted instantaneously from just a few data points, e.g. quickly incorporating expert feedback in the form of a few examples. In Table 4, we show that StructCBR serves the purpose of instantaneously improving model performance (+2.6 pts EM), while being roughly 50$\times$ faster than finetuning for a single epoch, and 5000$\times$ faster than finetuning for 100 epochs of 30 case examples from world_1 schema. Finetuning required atleast 10 epochs to outperform StructCBR's accuracy. Each epoch involved four parameter update steps of batch-size 8. We note that applying StructCBR over SmBoP incurs a small increase ($\sim$1.2$\times$) in inference time per example. Overall, we observe that on three out of five schemas StructCBR instantly offers more than 50% of gains achieved by finetuning for 100 epochs, and 43% of gains on average across all schemas (Table A2 in appendix). This establishes StructCBR as a method for instantly utilizing available case examples for fast model adaptation until the next cycle of finetuning becomes affordable.

Adaptation Method	Time(s)↓	EM%↑
SmBOP (Unadapted)	0.0	48.3
StructCBR	0.1	50.6
Finetuning (1 epochs)	5.0	48.3
Finetuning (2 epochs)	10.0	47.5
Finetuning (5 epochs)	25.0	48.3
Finetuning (10 epochs)	50.0	50.2
Finetuning (20 epochs)	100.0	50.9
Finetuning (100 epochs)	500.0	52.1

Table 4: Comparing adaptation time and EM accuracy of StructCBR and finetuning for different number of epochs on 30 cases of world_1 schema. We report wall clock times in seconds. All the numbers were averaged over 3 runs. Finetuning took atleast 500x more time (10 epochs) to achieve EM gains that are almost instantly (0.1s) achieved by StructCBR

Conclusion and Future Work

We presented StructCBR, a method for instant adaptation of Text-to-SQL models without finetuning. We show that utilzing case examples in a more structured way via sub-tree level look-ups offers better performance in comparison to the standard method of concatenating case examples with input text into a Seq2Seq encoder. We find that explicitly learning to perform memory look-ups provides larger gains in comparison to look-ups using a pre-trained model. Finally, we show that StructCBR enables much faster model adaptation in comparison to finetuning, potentially allowing instantaneous adaptation to expert feedback provided in form of a few examples. We propose StructCBR as a faster alternative to model adaptation, until the next finetuning cycle is deemed feasible. Despite its speed, there remains an accuracy gap between StructCBR and sufficient finetuning, which might be narrowed by more sophisticated similarity networks. Our exploration of StructCBR focused only on the Text-to-SQL task. In future we wish to explore the effectiveness of StructCBR for other semantic parsing tasks.

References

Atzeni, M.; Dhuliawala, S. Z.; Murugesan, K.; and SACHAN, M. 2022. Case-based Reasoning for Better Generalization in Text-Adventure Games. In *International Conference on Learning Representations*.

Das, R.; Godbole, A.; Monath, N.; Zaheer, M.; and McCallum, A. 2020. Probabilistic Case-based Reasoning for Open-World Knowledge Graph Completion. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, 4752–4765.

Das, R.; Zaheer, M.; Thai, D.; Godbole, A.; Perez, E.; Lee, J.-Y.; Tan, L.; Polymenakos, L.; and McCallum, A. 2021. Case-based Reasoning for Natural Language Queries over Knowledge Bases. *arXiv preprint arXiv:2104.08762*.

Gardner, M.; Grus, J.; Neumann, M.; Tafjord, O.; Dasigi, P.; Liu, N. F.; Peters, M.; Schmitz, M.; and Zettlemoyer, L. 2018. AllenNLP: A Deep Semantic Natural Language Processing Platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, 1–6. Melbourne, Australia: Association for Computational Linguistics.

Guo, J.; Zhan, Z.; Gao, Y.; Xiao, Y.; Lou, J.-G.; Liu, T.; and Zhang, D. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 4524–4535. Florence, Italy: Association for Computational Linguistics.

Gupta, V.; Shrivastava, A.; Sagar, A.; Aghajanyan, A.; and Savenkov, D. 2021. RETRONLU: Retrieval Augmented Task-Oriented Semantic Parsing. *arXiv preprint arXiv:2109.10410*.

Hashimoto, T. B.; Guu, K.; Oren, Y.; and Liang, P. S. 2018. A retrieve-and-edit framework for predicting structured outputs. *Advances in Neural Information Processing Systems*, 31.

Hazoom, M.; Malik, V.; and Bogin, B. 2021. Text-to-SQL in the Wild: A Naturally-Occurring Dataset Based on Stack Exchange Data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, 77–87. Online: Association for Computational Linguistics.

Hossain, N.; Ghazvininejad, M.; and Zettlemoyer, L. 2020. Simple and effective retrieve-edit-rerank text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2532–2538.

Huang, X. S.; Perez, F.; Ba, J.; and Volkovs, M. 2020. Improving Transformer Optimization Through Better Initialization. In III, H. D.; and Singh, A., eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 4475–4483. PMLR.

Khandelwal, U.; Fan, A.; Jurafsky, D.; Zettlemoyer, L.; and Lewis, M. 2021. Nearest Neighbor Machine Translation. In *International Conference on Learning Representations*.

Khandelwal, U.; Levy, O.; Jurafsky, D.; Zettlemoyer, L.; and Lewis, M. 2020. Generalization through Memorization: Nearest Neighbor Language Models. In *International Conference on Learning Representations*.

Lee, C.-H.; Polozov, O.; and Richardson, M. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2261–2273. Online: Association for Computational Linguistics.

Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; and Stoyanov, V. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Pasupat, P.; Zhang, Y.; and Guu, K. 2021. Controllable Semantic Parsing via Retrieval Augmentation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 7683–7698.

Popescu, A.-M.; Etzioni, O.; and Kautz, H. 2003. Towards a Theory of Natural Language Interfaces to Databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, 149–157.

Rubin, O.; and Berant, J. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 311–324.

Scholak, T.; Li, R.; Bahdanau, D.; de Vries, H.; and Pal, C. 2021. DuoRAT: Towards Simpler Text-to-SQL Models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1313–1321.

Scholak, T.; Schucher, N.; and Bahdanau, D. 2021. PICARD - Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Shaw, P.; Uszkoreit, J.; and Vaswani, A. 2018. Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, 464–468. New Orleans, Louisiana: Association for Computational Linguistics.

Socher, R.; Chen, D.; Manning, C. D.; and Ng, A. 2013. Reasoning with neural tensor networks for knowledge base completion. *Advances in neural information processing systems*, 26.

Suhr, A.; Chang, M.-W.; Shaw, P.; and Lee, K. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 8372–8388.

Tang, L. R.; and Mooney, R. J. 2000. Automated Construction of Database Interfaces: Intergrating Statistical and Relational Learning for Semantic Parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 133–141. Hong Kong, China: Association for Computational Linguistics.

Wang, B.; Shin, R.; Liu, X.; Polozov, O.; and Richardson, M. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the*

58th Annual Meeting of the Association for Computational Linguistics, 7567–7578.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; Davison, J.; Shleifer, S.; von Platen, P.; Ma, C.; Jernite, Y.; Plu, J.; Xu, C.; Scao, T. L.; Gugger, S.; Drame, M.; Lhoest, Q.; and Rush, A. M. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. Online: Association for Computational Linguistics.

Xu, P.; Kumar, D.; Yang, W.; Zi, W.; Tang, K.; Huang, C.; Cheung, J. C. K.; Prince, S. J.; and Cao, Y. 2021. Optimizing Deeper Transformers on Small Datasets. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2089–2102. Online: Association for Computational Linguistics.

Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; et al. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3911–3921.

Zelle, J. M.; and Mooney, R. J. 1996. Learning to Parse Database Queries using Inductive Logic Programming. In *AAAI/IAAI*, 1050–1055. Portland, OR: AAAI Press/MIT Press.